Enriching an effect calculus with linear types

Egger, J., Møgelberg, R. E., & Simpson, A. (2009). Enriching an effect calculus with linear types. In Proceedings of the 23rd CSL international conference and 18th EACSL Annual conference on Computer science logic. (pp. 240-254). (Lecture Notes in Computer Science). Berlin, Heidelberg: Springer Verlag. doi: 10.1007/978-3-642-04027-6

Enriching an Effect Calculus with Linear Types

Jeff Egger [⋆1], Rasmus Ejlers Møgelberg [⋆⋆2], and Alex Simpson [⋆1]

[1] LFCS, School of Informatics, University of Edinburgh, Scotland, UK
[2] IT University of Copenhagen, Copenhagen, Denmark

Abstract. We define an *enriched effect calculus* by extending a type theory for computational effects with primitives from linear logic. The new calculus provides a formalism for expressing linear aspects of computational effects; for example, the linear usage of imperative features such as state and/or continuations.

Our main syntactic result is the conservativity of the enriched effect calculus over a basic *effect calculus* without linear primitives (closely related to Moggi's *computational metalanguage*, Filinski's *effect PCF* and Levy's *call-by-push-value*). The proof of this syntactic theorem makes essential use of a category-theoretic semantics, whose study forms the second half of the paper.

Our semantic results include soundness, completeness, the initiality of a syntactic model, and an embedding theorem: every model of the basic effect calculus fully embeds in a model of the enriched calculus. The latter means that our enriched effect calculus is applicable to arbitrary computational effects, answering in the positive a question of Benton and Wadler (LICS 1996).

1 Introduction

The *computational metalanguage* was proposed by Moggi [14] as a general metalanguage for ascribing semantics to programming languages with effects, building on his own idea that computational effects can be encapsulated by the mathematical structure of a *strong monad* [13]. The metalanguage extends the simply-typed λ-calculus with a new type constructor T, where TA represents a type of all computations that produce values of type A. Semantically, T is interpreted as a strong monad that captures the effects that computations may exhibit.

In [4], Benton and Wadler identify a close connection between Moggi's computational metalanguage and Girard's intuitionistic linear logic (ILL) [7]. They show that every model of ILL can be reconstrued as a model of the computational metalanguage, and this determines an interpretation of the computational metalanguage within ILL. However, the models of the computational metalanguage that arise from models of ILL are very special ones: their monads are *commutative*. This means that the interpretation of the computational metalanguage in

⋆ Research supported by an EPSRC research grant EP/F042043/1.
⋆⋆ Research supported by the Danish Agency for Science, Technology and Innovation.

ILL validates an equation that is not always true for effects: the equation that asserts the insensitivity of computational effects to their order of execution.

Many computational effects are *not* commutative, for example: exceptions, state, input/output, and continuations. In [4, §8], Benton and Wadler write:

> *"We do not know if it is possible to define a non-commutative linear calculus which corresponds to a wider class of monad models."*

In this paper, we present such a calculus, the *enriched effect calculus*.

The enriched effect calculus can be viewed as an extension of Moggi's metalanguage with a judicious selection of type constructors from linear logic. We envisage that this calculus will be applicable to computational scenarios in which the manipulation of computational effects adheres to a discipline of linearity. For example, the usual *state monad* $S \to ((-) \times S)$ (where S is an object of *states*) has a linear counterpart $S \multimap (!(-) \otimes S)$, which accounts for the fact that state (unlike values) can neither be duplicated nor discarded, cf. [15]. Similarly, the *continuations monad* $((-) \to R) \to R$ (where R is a *result type*) has a linear version $((-) \to R) \multimap R$, which enforces the *linear usage* of continuations, a discipline that is ubiquitous amongst structured forms of control [3].

While examples such as the above can be formulated in ILL itself, we believe our enriched effect calculus to be the natural home for them. Indeed, the enriched effect calculus has two main advantages over ILL. First, it is weaker than ILL, and hence applicable more widely (models of ILL are a strict subset of models of the enriched effect calculus). Second, the tight connection between the enriched effect calculus and Moggi's computational metalanguage, which we establish in this paper, makes the former a natural vehicle for formalising the general phenomenon of interactions between linearity and effects, including the *linear usage of effects* [9]. Detailed applications of the enriched effect calculus to such examples will appear elsewhere, starting with a study of linearly-used continuations in a follow-up paper [5].

In this paper, our enriched effect calculus is defined as an extension of a basic *effect calculus*, which is presented in Section 2. The basic calculus is a simple (and essentially standard) extension of Moggi's computational metalanguage with a notion of *computation type*, as used in Filinski's *effect PCF* [6] and in Levy's *call-by-push-value* [11]. The *enriched effect calculus* is presented in Section 3. There we state our main syntactic theorem: the enriched calculus conservatively extends the basic calculus. Thus the addition of the new linear logic connectives does not interfere with the existing type constructors of the basic calculus.

The second half of the paper, starting with Section 4, is devoted to category-theoretic models. As models of the basic effect calculus we take (an appropriate version of) Levy's *adjunction models* [12], which are the natural models for calculi based on computation types. As models of the enriched effect calculus, we take adjunction models with the extra structure needed to model the linear connectives. This structure is formulated in terms of *enriched category theory* [10]. Soundness and completeness are addressed in Section 4. In Section 5, morphisms of models are defined and an initiality property is formulated for a syntactic

model. Finally, in Section 6, we show that every adjunction model fully embeds in a model of the enriched effect calculus. It is this theorem that answers Benton and Wadler's (implicit) question quoted above: the enriched effect calculus is compatible with *any* monad model (any such can be presented as a Levy adjunction model). Furthermore, the embedding theorem is also used to prove the syntactic conservativity of the enriched effect calculus over the basic calculus.

2 A basic effect calculus

Moggi's *computational metalanguage*, [14], extends the simply-typed λ-calculus with new types TA, which type computations (possibly with effects) that produce values of type A. The new type has an associated "let" operator, which performs the *Kleisli extension* of a map $A \to TB$ to a map $TA \to TB$. This can be seen as a restricted form of elimination rule for the type TA. Filinski [6] generalises this elimination rule to apply to a wider class of "target" types than those of the form TB, and develops a calculus for this based on classifying such types as special *computation types* within a broader class of *value types*. Such a generalisation is useful for interpreting call-by-name languages. Its importance has been thoroughly established by Levy, whose *call-by-push-value (CBPV)* paradigm [11] is based on the distinction between computation and value types.

Guided by the above, we define our *effect calculus* as an extension of the simply-typed λ-calculus with a new type constructor, following Moggi, and with a division of types into value types and computation types, following Filinski and Levy. Because we have two classes of types, we assume two classes of type constants. We use $\alpha, \beta, \ldots$ to range over a set of *value type constants*, and $\underline{\alpha}, \underline{\beta}, \ldots$ to range over a disjoint set of *computation type constants*. We then use $\mathsf{A}, \mathsf{B}, \ldots$ to range over *value types*, and $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \ldots$ to range over *computation types*, which are specified by the grammar below:

$$\mathsf{A} ::= \alpha \mid \underline{\alpha} \mid 1 \mid \mathsf{A} \times \mathsf{B} \mid \mathsf{A} \to \mathsf{B} \mid \,!\mathsf{A}$$
$$\underline{\mathsf{A}} ::= \underline{\alpha} \mid 1 \mid \underline{\mathsf{A}} \times \underline{\mathsf{B}} \mid \mathsf{A} \to \underline{\mathsf{B}} \mid \,!\mathsf{A}$$

By this definition, every computation type is also a value type.

Our notation for type constructors is standard, except that we use the linear exponential notation $!\mathsf{A}$ for Moggi's monadic type TA. (The reasons for this non-standard choice will transpire later.) We follow Filinski in making computation types a subclass of value types. Levy, in contrast, keeps computation types and value types separate. He has an operator F that turns a value type A into a computation type $F\mathsf{A}$, and conversely an operator U that maps a computation type $\underline{\mathsf{A}}$ to a value type $U\underline{\mathsf{A}}$. Levy's type $F\mathsf{A}$ corresponds to $!\mathsf{A}$ in our syntax, and his type $U\underline{\mathsf{A}}$ is simply $\underline{\mathsf{A}}$ itself. Three reasons for our choice of omitting U and subsuming computation types as value types are: the streamlined syntax leads to a very economical type system (see below); there is no loss of information, since one can establish an equivalence between the two systems; and also the term syntax is not cluttered with (inferable) conversions between values and

computations. However, for the purposes of the present paper, the main benefit of our formulation is that our syntax provides a transparent foundation for the extension with linear logic connectives in Section 3.

We mention two other differences from CBPV, as presented in [11]. First, because we are building on the simply-typed λ-calculus, we include a full function space between value types, whereas Levy only includes function spaces as computation types, and, as in the present paper, these are required to have computation type codomains. Second, Levy includes sums of value types as a basic construct, whereas, largely for space reasons, we have omitted them. Since either of these differences could be easily circumvented by making evident minor alterations to our type system (or to CBPV), we believe it is correct to view our effect calculus as essentially CBPV, albeit with a different syntax.

To ease the transition to the linear calculus, we shall build a notion of linearity directly into the typing judgements of the basic effect calculus. This notion has an intuitive motivation. Following Levy [11], we view value types as typing *values*, which are static entities, and we view computation types as typing *computations*, which are dynamic entities. If a term t has computation type, and contains a parameter z of computation type then there is a natural notion of t depending linearly on z: the execution of the computation t contains within it exactly one execution of the computation z. Since computations may perform arbitrary effects including nontermination, such a linear dependency can only hold in general if the execution of z is the first subcomputation performed in the execution of t. (If, for example, a computation that diverges were due to be performed before z then z might never be executed.) Thus we may rephrase, t depends linearly on z if the execution of the computation t begins with the execution of the computation z. Accordingly, we have arrived at a notion of t depending linearly on z that is similar in spirit to saying that $t[z]$ is an evaluation context. The situation for value types is fundamentally different. Since values are static, they are reasonably considered as pervasive entities that might be used any number of times. Accordingly, there is no natural notion of a term t depending linearly on a parameter x of value type.

The above discussion is intended to give informal motivation for considering type rules for the effect calculus based on two typing judgements:

$$\text{(i)}\ \ \Gamma \mid - \vdash t\colon \mathsf{A}$$
$$\text{(ii)}\ \ \Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}\ ,$$

where Γ is a context of value-type assignments to variables. On the right of Γ is a *stoup* (following the terminology of [8]), which may either be empty, as in the case of judgement (i), or may consist of a unique type assignment $z\colon \underline{\mathsf{A}}$, in which case the type on the right of the turnstyle is also required to be a computation type, as in (ii). The purpose of judgement (i) is merely to assert that t has value type A in (value) context Γ. Judgement (ii) asserts that t is a computation of type $\underline{\mathsf{B}}$ (in context Γ) which depends linearly on the computation z of type $\underline{\mathsf{A}}$. Note how these two judgement forms correspond to the informal discussion of linearity given above. This discussion also provides some intuitive

$$\frac{}{\Gamma,\, x\colon \mathsf{A} \mid - \vdash x\colon \mathsf{A}} \qquad \frac{}{\Gamma \mid z\colon \underline{\mathsf{A}} \vdash z\colon \underline{\mathsf{A}}} \qquad \frac{}{\Gamma \mid \Delta \vdash *\colon 1}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \qquad \Gamma \mid \Delta \vdash u\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \langle t, u\rangle\colon \mathsf{A} \times \mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathsf{fst}(t)\colon \mathsf{A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{A} \times \mathsf{B}}{\Gamma \mid \Delta \vdash \mathsf{snd}(t)\colon \mathsf{B}}$$

$$\frac{\Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{A}.\, t\colon \mathsf{A} \to \mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon \mathsf{A} \to \mathsf{B} \qquad \Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid \Delta \vdash s(t)\colon \mathsf{B}}$$

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid - \vdash \mathsf{!}\, t\colon \mathsf{!A}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon \mathsf{!A} \qquad \Gamma,\, x\colon \mathsf{A} \mid - \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathsf{let}\, \mathsf{!}\, x\, \mathsf{be}\, t\, \mathsf{in}\, u\colon \underline{\mathsf{B}}}$$

Fig. 1. Typing rules for the effect calculus

motivation for the restriction of the linear context to a stoup containing at most one variable of computation type. Since a linearly-used parameter z (necessarily of computation type) must be executed first in the execution of t, it is natural that just one variable can enjoy this property.

The typing rules are given in Figure 1. In them, Δ ranges over an arbitrary (possibly empty) stoup, and the rules are only applicable in the case of typing judgements that conform to (i) or (ii) above. The positioning of the stoups Δ in the rules can be understood in terms of the intuitive definition of linearity given above. For example, the evaluation behaviour of the terms associated with $!A$ types can be understood following Moggi [13, 14]. In the introduction rule, the term $!t$ represents the trivial computation that immediately returns the value t of type A. There is no space in this for any linear dependency on a subcomputation z. In the elimination rule, the term $\mathsf{let}\,!\,x\,\mathsf{be}\,t\,\mathsf{in}\,u$ first evaluates the computation t, binds the result (if any) to x and then proceeds to evaluate the computation u. Clearly if z is evaluated first within t then it is also evaluated first within $\mathsf{let}!\,x\mathsf{be}t\;\mathsf{in}u$, and this justifies the positioning of Δ in the rule. Observe, however, that the following variation on the rule is not legitimised by our interpretation of linearity, and hence is not included in the calculus.

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{!A} \qquad \Gamma,\, x\colon \mathsf{A} \mid \Delta \vdash u\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathsf{let}\, \mathsf{!}\, x\, \mathsf{be}\, t\, \mathsf{in}\, u\colon \underline{\mathsf{B}}} \tag{1}$$

The problem here is that any z in Δ is evaluated as part of u, and this occurs only after t has been evaluated first. Similar explanations can be given to the other rules. They rely on giving the products a lazy interpretation: components are only evaluated once projected out. (So, e.g., the linearity in the rule for 1 is correct because 1 is the empty product and $*$ can never be projected.)

Modulo the differences from CBPV already mentioned above (abitrary function spaces, no sum types) the type system discussed is essentially just a concise reformulation of CBPV with *complex stacks*, as found in [11, 12]. In particular,

$$
\begin{array}{ll}
\Gamma \mid \Delta \vdash t = * \colon 1 & \text{if } \Gamma \mid \Delta \vdash t \colon 1 \\
\Gamma \mid \Delta \vdash \mathsf{fst}(\langle t, u \rangle) = t \colon \mathsf{A} & \text{if } \Gamma \mid \Delta \vdash t \colon \mathsf{A} \text{ and } \Gamma \mid \Delta \vdash t \colon \mathsf{B} \\
\Gamma \mid \Delta \vdash \mathsf{snd}(\langle t, u \rangle) = u \colon \mathsf{B} & \text{if } \Gamma \mid \Delta \vdash t \colon \mathsf{A} \text{ and } \Gamma \mid \Delta \vdash t \colon \mathsf{B} \\
\Gamma \mid \Delta \vdash \langle \mathsf{fst}(t), \mathsf{snd}(t) \rangle = t \colon \mathsf{A} \times \mathsf{B} & \text{if } \Gamma \mid \Delta \vdash t \colon \mathsf{A} \times \mathsf{B} \\
\Gamma \mid \Delta \vdash (\lambda x \colon \mathsf{A}.\, t)(u) = t[u/x] \colon \mathsf{B} & \text{if } \Gamma, x \colon \mathsf{A} \mid \Delta \vdash t \colon \mathsf{B} \text{ and } \Gamma \mid - \vdash u \colon \mathsf{A} \\
\Gamma \mid \Delta \vdash \lambda x \colon \mathsf{A}.\, (t(x)) = t \colon \mathsf{A} \to \mathsf{B} & \text{if } \Gamma \mid \Delta \vdash t \colon \mathsf{A} \to \mathsf{B} \text{ and } x \notin \Gamma, \Delta \\
\Gamma \mid - \vdash \mathsf{let}\ !x\ \mathsf{be}\ !t\ \mathsf{in}\ u = u[t/x] \colon \underline{\mathsf{B}} & \text{if } \Gamma \mid - \vdash t \colon \mathsf{A} \text{ and } \Gamma, x \colon \mathsf{A} \mid - \vdash u \colon \underline{\mathsf{B}} \\
\Gamma \mid \Delta \vdash \mathsf{let}\ !x\ \mathsf{be}\ t\ \mathsf{in}\ u[!\,x/y] = u[t\,/\,y] \colon \underline{\mathsf{B}} & \text{if } \Gamma \mid \Delta \vdash t \colon\ !\mathsf{A} \text{ and } \Gamma \mid y \colon !\mathsf{A} \vdash u \colon \underline{\mathsf{B}}
\end{array}
$$

Fig. 2. Equality rules for the effect calculus

our judgement form $\Gamma \mid - \vdash t \colon \mathsf{A}$ corresponds to Levy's $\Gamma \vdash^v M \colon \mathsf{A}$, and our judgement $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ corresponds to Levy's $\Gamma \mid \underline{\mathsf{A}} \vdash^k K \colon \underline{\mathsf{B}}$.

Once again, our formulation has been chosen both for its economy and to make the extension with linear connectives transparent. Indeed, we have stayed close to linear logic notation (the exception is the use of $\times$ for product rather than the usual linear &), and our typing rules are simply restrictions, from an arbitrary linear context to a stoup, of the rules for ILL in [1]. This, in part, motivates the nonstandard use of $!\mathsf{A}$ instead of $T\mathsf{A}$. The one mismatch here is the illegitimate rule (1) above, which is valid in the context of ILL. In spite of this mismatch, it is our belief that the extension of the effect calculus with linear primitives presented below will make it clear that the overlap with linear logic is so strong that the linear notation is helpful more than it is misleading.

In Figure 2, we present rules for equalities between typed terms in the effect calculus. They are to be considered in addition to the expected (typed) congrence and α-equivalence rules.

3 The enriched effect calculus

The enriched effect calculus is obtained by adding a selection of type constructions from linear logic to the effect calculus. As befits the setting, this needs to be done respecting both the distinction between value and computation types, and the interpretation of linearity as a concept related to the latter.

We start with linear function types. In our setting, we have a notion of linearity between computation types only. Thus we add a type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$, internalising the linear dependency of judgements $\Gamma \mid z \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$. In order to have a calculus with a sufficiently wide collection of models (all monad models) it seems essential not to assume that $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is a computation type in general. This restriction is also natural if one thinks of a linear dependency $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a transformation of computations (much like an evaluation context) rather than a computation itself. The same restriction fits naturally with the stoup-based typing judgements,

$$\frac{\Gamma \mid z{:}\underline{\mathsf{A}} \vdash t{:}\,\underline{\mathsf{B}}}{\Gamma \mid - \vdash \underline{\lambda} z{:}\underline{\mathsf{A}}.\, t{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid - \vdash s{:}\,\underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \mid \Delta \vdash t{:}\,\underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash s[t]{:}\,\underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid - \vdash t{:}\,\mathsf{A} \quad \Gamma \mid \Delta \vdash u{:}\,\underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash\, !t \otimes u{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash s{:}\,!\mathsf{A} \otimes \underline{\mathsf{B}} \quad \Gamma, x{:}\mathsf{A} \mid z{:}\underline{\mathsf{B}} \vdash t{:}\,\underline{\mathsf{C}}}{\Gamma \mid \Delta \vdash \text{let } !x \otimes z \text{ be } s \text{ in } t{:}\,\underline{\mathsf{C}}}$$

$$\frac{\Gamma \mid \Delta \vdash t{:}\,\underline{0}}{\Gamma \mid \Delta \vdash \underline{\text{image}}(t){:}\,\underline{\mathsf{A}}} \qquad \frac{\Gamma \mid \Delta \vdash t{:}\,\underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash \underline{\text{inl}}(t){:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash t{:}\,\underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \underline{\text{inr}}(t){:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid \Delta \vdash s{:}\,\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \quad \Gamma \mid x{:}\underline{\mathsf{A}} \vdash t{:}\,\underline{\mathsf{C}} \quad \Gamma \mid y{:}\underline{\mathsf{B}} \vdash u{:}\,\underline{\mathsf{C}}}{\Gamma \mid \Delta \vdash \underline{\text{case}}\, s \,\text{of}\, (\underline{\text{inl}}(x).\, t; \underline{\text{inr}}(y).\, u){:}\,\underline{\mathsf{C}}}$$

Fig. 3. Additional typing rules for the enriched effect calculus.

since allowing a linear function to depend linearly on another parameter would lead to typing rules involving multivariable linear contexts.

In linear logic, linear function space is intimately connected to the tensor product $\otimes$, which normally internalises the comma separating types in the linear context of a typing judgement. In our stoup-based system, there is at most one type in the linear context (the stoup), and so it seems awkward to implement the usual symmetric $\otimes$. Similarly, it is also difficult to find an appropriate $\otimes$ operation in a sufficiently general class of models. What does work, both syntactically and semantically, is an asymmetric version: for any value type A and computation type $\underline{\mathsf{B}}$, we include a new computation (and hence value) type $!\mathsf{A} \otimes \underline{\mathsf{B}}$. Note that this is the application of a single primitive binary constructor. The hybrid notation is chosen to maintain consistency with linear logic.

Finally, we include linear coproducts of computation types, $\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}$ and $\underline{0}$.

The resulting *enriched effect calculus* has types defined by extending the grammar for value and computation types of the effect calculus with the following additional type constructors.

$$\begin{aligned} \mathsf{A} &::= \ldots \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \mid !\mathsf{A} \otimes \underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \\ \underline{\mathsf{A}} &::= \ldots \mid !\mathsf{A} \otimes \underline{\mathsf{B}} \mid \underline{0} \mid \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \ . \end{aligned}$$

The judgement forms for the enriched effect calculus are exactly as for the effect calculus (now using the extended range of types). The additional typing rules are presented in Figure 3. Again, they can be seen to be restrictions of standard intuitionistic linear logic rules, as in [1]. The equality theory on terms is extended by the rules in Figure 4.

The restriction that $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is a value type, and the lack of a symmetric tensor have consequences on expressivity that may, at first, seem drastic. An obvious restriction is that linear function space does not iterate: neither $\underline{\mathsf{A}} \multimap (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}})$ nor $(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}}$ are allowed. However, it is possible to interleave linear function space with full function space. For example, $\underline{\mathsf{A}} \multimap (\mathsf{B} \to \underline{\mathsf{C}})$ is a value type, and

$$
\begin{array}{ll}
\Gamma \mid \Delta \vdash (\underline{\lambda} x\colon \underline{\mathsf{A}}.\, t)[u] = t[u/x] \colon \underline{\mathsf{B}} & \text{if } \Gamma \mid x\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}} \text{ and } \Gamma \mid \Delta \vdash u\colon \underline{\mathsf{A}} \\
\Gamma \mid - \vdash \underline{\lambda} x\colon \underline{\mathsf{A}}.\, (t[x]) = t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} & \text{if } \Gamma \mid - \vdash t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \text{ and } x \notin \Gamma \\
\Gamma \mid \Delta \vdash \text{let } !x \otimes y \text{ be } !t \otimes s \text{ in } u = u[t, s/x, y] \colon \underline{\mathsf{C}} & \text{if } \Gamma \mid - \vdash t\colon \mathsf{A} \text{ and } \Gamma \mid \Delta \vdash s\colon \underline{\mathsf{B}} \\
 & \quad \text{and } \Gamma, x\colon \mathsf{A} \mid y\colon \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}} \\
\Gamma \mid \Delta \vdash \text{let } !x \otimes y \text{ be } t \text{ in } u[!x \otimes y/z] = u[t/z] \colon \underline{\mathsf{C}} & \text{if } \Gamma \mid \Delta \vdash t\colon !\mathsf{A} \otimes \underline{\mathsf{B}} \text{ and } \Gamma \mid z\colon !\mathsf{A} \otimes \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}} \\
\Gamma \mid x\colon \underline{0} \vdash t = \underline{\text{image}}(x)\colon \underline{\mathsf{A}} & \text{if } \Gamma \mid x\colon \underline{0} \vdash t\colon \underline{\mathsf{A}} \\
\Gamma \mid \Delta \vdash \underline{\text{case}}\, \underline{\text{inl}}(t) \text{ of } (\underline{\text{inl}}(x).\, u; \underline{\text{inr}}(y).\, u') & \text{if } \Gamma \mid x\colon \underline{\mathsf{A}} \vdash u\colon \underline{\mathsf{C}} \text{ and } \Gamma \mid y\colon \underline{\mathsf{B}} \vdash u'\colon \underline{\mathsf{C}} \\
\qquad = u[t/x] : \underline{\mathsf{C}} & \quad \text{and } \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}} \\
\Gamma \mid \Delta \vdash \underline{\text{case}}\, \underline{\text{inr}}(t) \text{ of } (\underline{\text{inl}}(x).\, u; \underline{\text{inr}}(y).\, u') & \text{if } \Gamma \mid x\colon \underline{\mathsf{A}} \vdash u\colon \underline{\mathsf{C}} \text{ and } \Gamma \mid y\colon \underline{\mathsf{B}} \vdash u'\colon \underline{\mathsf{C}} \\
\qquad = u'[t/y] : \underline{\mathsf{C}} & \quad \text{and } \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{B}} \\
\Gamma \mid \Delta \vdash \underline{\text{case}}\, t \text{ of } (\underline{\text{inl}}(x).u[\underline{\text{inl}}(x)/z]; \underline{\text{inr}}(y).\, u[\underline{\text{inr}}(y)/z]) & \\
\qquad = u[t/z] : \underline{\mathsf{C}} & \text{if } \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \text{ and } \Gamma \mid z\colon \underline{\mathsf{A}} \oplus \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}}
\end{array}
$$

Fig. 4. Additional equality rules for the enriched effect calculus.

$(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \to \underline{\mathsf{C}}$ is a computation (and hence value) type. Thus, for example, the linearly-used continuations monad, $((-) \to \underline{R}) \multimap \underline{R})$ is implementable as a monad on value types, for any computation type $\underline{R}$ of results. Likewise, the linearly-used state monad $\underline{S} \multimap (!(-) \otimes \underline{S})$, which makes use of the asymmetric tensor, is implementable for any computation type $\underline{S}$ of states. Such examples will be treated in detail in future papers, starting with [5].

Many of the familiar laws of linear logic transfer to the enriched effect calculus, insofar as they can be expressed. For example:

$$
\begin{array}{rcll}
\mathsf{A} \to \underline{\mathsf{B}} & \cong & !\mathsf{A} \multimap \underline{\mathsf{B}} & \text{as value types} \\
!\mathsf{A} \otimes !\mathsf{B} & \cong & !(\mathsf{A} \times \mathsf{B}) & \text{as computation types} \\
!\mathsf{A} \otimes (\underline{\mathsf{B}} \oplus \underline{\mathsf{C}}) & \cong & (!\mathsf{A} \otimes \underline{\mathsf{B}}) \oplus (!\mathsf{A} \otimes \underline{\mathsf{C}}) & \text{as computation types} \\
(!\mathsf{A} \otimes \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}} & \cong & \mathsf{A} \to (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}}) & \\
 & \cong & \underline{\mathsf{B}} \multimap (\mathsf{A} \to \underline{\mathsf{C}}) & \text{as value types} \\
!1 \otimes \underline{\mathsf{A}} & \cong & \underline{\mathsf{A}} & \text{as computation types,}
\end{array}
$$

where the isomorphisms between computation types are themselves linear. The above laws demonstrate that our linear connectives behave just in the way linear logic leads us to expect they should. We take this as justification for our decision to adopt linear logic notation, including the choice of replacing $T\mathsf{A}$ with $!\mathsf{A}$.

Our main theorem about the enriched effect calculus is that the addition of the new linear type constructions is conservative over the basic effect calculus.

Theorem 1 (Conservativity). *Suppose Γ, Δ and A contain only types from the basic effect calculus.*

1. *If* $\Gamma \mid \Delta \vdash u \colon \mathsf{A}$ *is typable in the enriched effect calculus, then there exists a term* $\Gamma \mid \Delta \vdash t \colon \mathsf{A}$ *typable in the basic effect calculus such that* $\Gamma \mid \Delta \vdash t = u \colon \mathsf{A}$ *holds in the enriched effect calculus.*
2. *If* $\Gamma \mid \Delta \vdash s \colon \mathsf{A}$ *and* $\Gamma \mid \Delta \vdash t \colon \mathsf{A}$ *are typable in the basic effect calculus, and* $\Gamma \mid \Delta \vdash s = t \colon \mathsf{A}$ *holds in the enriched effect calculus, then* $\Gamma \mid \Delta \vdash s = t \colon \mathsf{A}$ *holds in the basic effect calculus.*

Statement 1 can be proved by a standard normalization argument, showing that every term t is provably equal to one for which every subterm is typed by a subtype of a type in the judgement typing t. The only proof we know of statement 2 is semantic, and will be given in Section 6.

4 Models

Our basic effect calculus is closely related to Levy's CBPV with stacks. Accordingly, the natural models are given by Levy's adjunction models [12]. While these are most simply presented as locally-indexed categories, see *op. cit.*, we use a definition based on enriched category theory, since this connects more easily with the models of the enriched effect calculus introduced below.

The idea of enriched category theory is to generalise the notion of category so that, rather than having *sets* of morphisms between pairs of objects, one has *hom-objects* given as objects of a specified "enriching" category $\mathbf{V}$. Thus a $\mathbf{V}$-*enriched category* (or $\mathbf{V}$-*category*), $\mathbf{C}$, is given by a collection of objects together with, for every pair of objects A, B, an associated object $\mathbf{C}(A, B)$ of the category $\mathbf{V}$. To make this work, the category $\mathbf{V}$ needs to be monoidal, and identity maps and an associative composition need to be specified on hom-objects. For space reasons, we refer readers to Kelly's book [10] for full definitions. Enriched category theory generalises ordinary category theory since an ordinary locally small category is just a **Set**-enriched category.

In this paper, we always consider enrichment with respect to categories $\mathbf{V}$ that are cartesian closed (we write B^A or $[A \to B]$ for functions spaces), and with the enriching monoidal structure as finite product. Any such category is self-enriched. Recall that for any small category $\mathbf{V}$, the Yoneda embedding $\mathbf{y}_{\mathbf{V}}$ fully and faithfully embeds $\mathbf{V}$ into the presheaf category $\widehat{\mathbf{V}} = \mathbf{Set}^{\mathbf{V}^{\mathrm{op}}}$, and since the latter is cartesian closed, $\mathbf{V}$ is $\widehat{\mathbf{V}}$-enriched.

We say that a $\mathbf{V}$-enriched category $\mathbf{C}$ has *($\mathbf{V}$-)powers* (Kelly writes *cotensors*) indexed by an object $A \in \mathbf{V}$ if, for each object $\underline{B}$ in $\mathbf{C}$, there exists an object $\underline{B}^A$ of $\mathbf{C}$ such that, for all objects $\underline{C} \in \mathbf{C}$, there is an isomorphism:

$$\xi_{A,\underline{B},\underline{C}} \colon \mathbf{C}(\underline{C}, \underline{B}^A) \longrightarrow \mathbf{C}(\underline{C}, \underline{B})^A \tag{2}$$

in $\mathbf{V}$, and such isomorphisms are $\mathbf{V}$-natural in $\underline{C}$. The dual property is that of having *($\mathbf{V}$-)copowers* (Kelly writes *tensors*) indexed by A: for each object $\underline{B}$ of $\mathbf{C}$, there must exist an object $A \cdot \underline{B}$ of $\mathbf{C}$ such that there are isomorphisms in $\mathbf{V}$,

$$\psi_{A,\underline{B},\underline{C}} \colon \mathbf{C}(A \cdot \underline{B}, \underline{C}) \longrightarrow \mathbf{C}(\underline{B}, \underline{C})^A \tag{3}$$

and again this family is $\mathbf{V}$-natural in $\underline{C}$.

An *enriched* adjunction $F \dashv U$ between $\mathbf{V}$-functors $F\colon \mathbf{D} \to \mathbf{C}$ and $U\colon \mathbf{C} \to \mathbf{D}$ requires the existence of isomorphisms in $\mathbf{V}$

$$\rho_{A,\underline{B}}\colon \mathbf{C}(F(A), \underline{B}) \longrightarrow \mathbf{D}(A, U(\underline{B})) \tag{4}$$

which are $\mathbf{V}$-natural in A and $\underline{B}$, cf. [10].

Definition 1. An *effect-calculus model* comprises a (small) cartesian closed category $\mathbf{V}$, together with a $\widehat{\mathbf{V}}$-enriched category $\mathbf{C}$ with: $\widehat{\mathbf{V}}$-enriched finite products, $\widehat{\mathbf{V}}$-powers indexed by representables, and a $\widehat{\mathbf{V}}$-adjunction $F \dashv U\colon \mathbf{C} \to \mathbf{V}$.

As mentioned above, effect-calculus models are an enriched-category formulation of Levy's *adjunction models* of CBPV [12], adapted to carry exactly the structure needed to model the effect calculus. Effect-calculus models are closely related to Moggi's monad-based metalanguage models [14]. Given an effect-calculus model, the composite UF carries the structure of a strong monad on $\mathbf{V}$. Conversely, given a strong monad T on a cartesian-closed category $\mathbf{V}$, the canonical adjunction $F \dashv U\colon \mathbf{V}^T \to \mathbf{V}$ between $\mathbf{V}$ and the category $\mathbf{V}^T$ of algebras for the monad, enriches to an effect-calculus model [12, Examples 4.9].

The idea behind effect-calculus models is that $\mathbf{V}$ is the category of value types and $\mathbf{C}$ the category of linear maps between computation types. The reason for requiring $\mathbf{C}$ to be enriched over $\widehat{\mathbf{V}}$ is to model judgements $\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ in non-empty contexts Γ, cf. [12]. In the *enriched* effect calculus, however, the presence of the linear function space $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a value type means that the hom-set $\mathbf{C}(\underline{A}, \underline{B})$ needs to live as an object of the category $\mathbf{V}$ of value types itself. This helps to make the definition of an *enriched-effect-calculus model* simpler and more natural than the notion of effect-calculus model.

Definition 2. An *enriched-effect-calculus model* comprises a cartesian closed category $\mathbf{V}$, together with a $\mathbf{V}$-enriched category $\mathbf{C}$ with: $\mathbf{V}$-enriched finite products and coproducts, powers, copowers, and a $\mathbf{V}$-adjunction $F \dashv U\colon \mathbf{C} \to \mathbf{V}$.

In an obvious way, each enriched-effect-calculus model can be reconstrued as an effect-calculus model. Although the converse, of course, does not hold, we shall show in Section 6 that every effect-calculus model can be fully embedded in an enriched-effect-calculus model.

A rich source of enriched-effect-calculus models is provided by models of ILL. Amongst the various formulations of such models, the most natural for our purposes is that of *linear/nonlinear* model [2], which consists of a cartesian-closed category $\mathbf{V}$ (the *intuitionistic category*), a symmetric monoidal closed category $\mathbf{C}$ (the *linear category*), and a symmetric monoidal adjunction $F \dashv G\colon \mathbf{C} \to \mathbf{V}$. To model the & and $\oplus$ operators of ILL one further requires $\mathbf{C}$ to have finite products and coproducts.

Proposition 1. *Every linear/nonlinear model in which the linear category has finite products and coproducts is an enriched-effect-calculus model.*

$$\begin{aligned}
\mathbf{V}[\![\underline{\alpha}]\!] &= U(\mathbf{C}[\![\underline{\alpha}]\!]) \\
\mathbf{V}[\![1]\!] &= 1 & \mathbf{C}[\![1]\!] &= 1 \\
\mathbf{V}[\![\mathsf{A} \times \mathsf{B}]\!] &= \mathbf{V}[\![\mathsf{A}]\!] \times \mathbf{V}[\![\mathsf{B}]\!] & \mathbf{C}[\![\underline{\mathsf{A}} \times \underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{A}}]\!] \times \mathbf{C}[\![\underline{\mathsf{B}}]\!] \\
\mathbf{V}[\![\mathsf{A} \to \mathsf{B}]\!] &= \mathbf{V}[\![\mathsf{A}]\!] \to \mathbf{V}[\![\mathsf{B}]\!] & \mathbf{C}[\![\mathsf{A} \to \underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{B}}]\!]^{\mathbf{V}[\![\mathsf{A}]\!]} \\
\mathbf{V}[\![!\mathsf{A}]\!] &= U(\mathbf{C}[\![!\mathsf{A}]\!]) & \mathbf{C}[\![!\mathsf{A}]\!] &= F(\mathbf{V}[\![\mathsf{A}]\!])
\end{aligned}$$

Fig. 5. Interpretation of effect-calculus types.

$$\begin{aligned}
\mathbf{V}[\![\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}]\!] &= \mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!], \mathbf{C}[\![\underline{\mathsf{B}}]\!]) \\
\mathbf{V}[\![!\mathsf{A} \otimes \underline{\mathsf{B}}]\!] &= U(\mathbf{V}[\![\mathsf{A}]\!] \cdot \mathbf{C}[\![\underline{\mathsf{B}}]\!]) & \mathbf{C}[\![!\mathsf{A} \otimes \underline{\mathsf{B}}]\!] &= \mathbf{V}[\![\mathsf{A}]\!] \cdot \mathbf{C}[\![\underline{\mathsf{B}}]\!] \\
\mathbf{V}[\![\underline{0}]\!] &= U(0) & \mathbf{C}[\![\underline{0}]\!] &= 0 \\
\mathbf{V}[\![\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}]\!] &= U(\mathbf{C}[\![\underline{\mathsf{A}}]\!] + \mathbf{C}[\![\underline{\mathsf{B}}]\!]) & \mathbf{C}[\![\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}]\!] &= \mathbf{C}[\![\underline{\mathsf{A}}]\!] + \mathbf{C}[\![\underline{\mathsf{B}}]\!]
\end{aligned}$$

Fig. 6. Interpretation of enriched-effect-calculus types.

Proof (outline). Since $\mathbf{V}$ and $\mathbf{C}$ are closed categories, they are self-enriched. The functor G, being monoidal, transports $\mathbf{C}$-enriched categories to $\mathbf{V}$-enriched categories by application of G to hom-objects, cf. [10, p.3]. In particular, we can apply this construction to $\mathbf{C}$ and get a $\mathbf{V}$-enriched category $G_\star(\mathbf{C})$ with hom-objects $G_\star(\mathbf{C})(\underline{A}, \underline{B}) = G(\underline{A} \multimap \underline{B})$. The symmetric monoidal adjunction $F \dashv G$ induces an enriched adjunction $F \dashv G \colon G_\star(\mathbf{C}) \to \mathbf{V}$. Powers and copowers can be defined respectively as $\underline{B}^A = F(A) \multimap \underline{B}$ and $A \cdot \underline{B} = F(A) \otimes \underline{B}$. □

We give a uniform treatment of the interpretation of the effect calculus in any effect-calculus model, and the interpretation of the enriched effect calculus in any enriched-effect-calculus model. In either case, a value type A is interpreted as an object $\mathbf{V}[\![\mathsf{A}]\!]$ of $\mathbf{V}$, and each computation type $\underline{\mathsf{A}}$ is interpreted as a pair $(\mathbf{C}[\![\underline{\mathsf{A}}]\!], s_{\underline{\mathsf{A}}})$ where: $\mathbf{C}[\![\underline{\mathsf{A}}]\!]$ is an object of $\mathbf{C}$, and $s_{\underline{\mathsf{A}}} \colon U(\mathbf{C}[\![\underline{\mathsf{A}}]\!]) \to \mathbf{V}[\![\underline{\mathsf{A}}]\!]$ is an isomorphism in $\mathbf{V}$. The interpretation is determined by specifying objects $\mathbf{V}[\![\alpha]\!] \in \mathbf{V}$ and $\mathbf{C}[\![\underline{\alpha}]\!] \in \mathbf{C}$, which we assume given. The remainder of the interpretation is defined in Figure 5, for both calculi, and also in Figure 6, for the remaining types of the enriched effect calculus. In the case of the effect calculus only, there is one notational simplification in Figure 5: in the definition of $\mathbf{C}[\![\mathsf{A} \to \underline{\mathsf{B}}]\!]$, the exponent of the power $\mathbf{C}[\![\underline{\mathsf{B}}]\!]^{\mathbf{V}[\![\mathsf{A}]\!]}$ should strictly be $\mathbf{y}(\mathbf{V}[\![\mathsf{A}]\!])$.

The non-trivial cases in the inductive definition of the isomorphism $s_{\underline{\mathsf{A}}}$ are the cases for unit type, product and function space. Since U is an enriched right adjoint, it preserves enriched limits and powers. The required isomorphisms are easily calculated from this preservation.

Terms are interpreted differently depending on whether they are typed with empty stoup or not. For $\Gamma \mid - \vdash t \colon \mathsf{A}$, the interpretation is a map in $\mathbf{V}$

$$\mathbf{V}[\![t]\!] \colon \mathbf{V}[\![\Gamma]\!] \to \mathbf{V}[\![\mathsf{A}]\!]$$

where $\mathbf{V}[\![\Gamma]\!]$ is the product of the interpretations of the types in Γ. A typing judgement $\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ is interpreted as a morphism:

$$\mathbf{C}[\![t]\!]\colon \mathbf{V}[\![\Gamma]\!] \to \mathbf{C}(\mathbf{C}[\![\underline{\mathsf{A}}]\!], \mathbf{C}[\![\underline{\mathsf{B}}]\!])\,.$$

For the effect calculus, this morphism is in $\widehat{\mathbf{V}}$ (and strictly its domain is $\mathbf{y}(\mathbf{V}[\![\Gamma]\!])$). For the enriched calculus it is in $\mathbf{V}$ itself. The interpretation of terms is defined by induction on the typing judgement. The details are omitted for lack of space.

Theorem 2. *Effect-calculus models are sound and complete with respect to the equational theory of Figure 2. Similarly, enriched-effect-calculus models are sound and complete with respect to the full equational theory of Figures 2 and 4.*

Proof (outline). As usual, completeness is proved by constructing a syntactic model. We consider the enriched effect calculus only. The syntactic category $\mathbf{V}_{\mathrm{Syn}}$ has as objects value types and as morphisms from A to B terms of the form $x\colon \mathsf{A} \mid - \vdash t\colon \mathsf{B}$ identified up to equality. The $\mathbf{V}_{\mathrm{Syn}}$-enriched category $\mathbf{C}_{\mathrm{Syn}}$ has as objects all computation types, with the $\mathbf{V}_{\mathrm{Syn}}$-object of morphisms from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ given by the value type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$. The functor F_{Syn} maps an object A to $!\mathsf{A}$, and the functor U_{Syn} maps $\underline{\mathsf{A}}$ to itself. The details are routine to verify. □

5 Morphisms of models

In this section, we define an appropriate notion of morphism between enriched-effect-calculus models.[1] The definition is subtle because it involves an unavoidable change of enrichment. Further intricacies are caused by the mathematically natural choice of requiring morphisms to preserve structure up to isomorphism rather than strictly.

Given two enriched models $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ and $F' \dashv U'\colon \mathbf{C}' \to \mathbf{V}'$, we need to consider what a structure-preserving morphism amounts to. Such a morphism must include a functor $S\colon \mathbf{V} \to \mathbf{V}'$ that preserves the cartesian-closed structure. The functor S determines a 2-functor $S_\star$ from the 2-category of $\mathbf{V}$-categories to that of $\mathbf{V}'$-categories, by application of S to hom-objects, cf. [10, p.3]. Hence, we obtain a $\mathbf{V}'$-enriched adjunction $S_\star(F) \dashv S_\star(U)\colon S_\star(\mathbf{C}) \to S_\star(\mathbf{V})$.

Next, observe that S determines a fully faithful $\mathbf{V}'$-functor S' from $S_\star(\mathbf{V})$ to $\mathbf{V}'$. This acts like S on objects, and its action on hom-objects is given by:

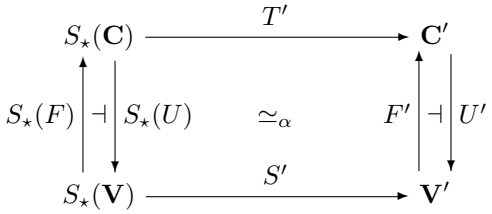
$$S_\star(\mathbf{V})(A, B) = S(\mathbf{V}(A, B)) = S[A \to_{\mathbf{V}} B] \cong [SA \to_{\mathbf{V}'} SB] = \mathbf{V}'(SA, SB)$$

using the preservation of cartesian-closedness by S.

A morphism of effect-calculus models must also include a component mapping $\mathbf{C}$ to $\mathbf{C}'$. Having performed the above change of enrichment, this is achieved by requiring a fully faithful $\mathbf{V}'$-functor $T'\colon S_\star(\mathbf{C}) \to \mathbf{C}'$ (the fully faithful property of T' amounts to the preservation of linear function spaces), together with a

[1] For lack of space, we do not define morphisms between effect-calculus models.

$\mathbf{V}'$-enriched natural isomorphism $\alpha\colon F'\,S' \Rightarrow T'\,S_\star(F)$ such that the thereby determined (via the adjunctions) $\mathbf{V}'$-natural transformation $\alpha'\colon S'\,S_\star(U) \Rightarrow U'\,T'$ (we call α' the *mate* of α) is also an isomorphism. Pictorially, we have:



The data (S', T', α) constitute a $\mathbf{V}'$-enriched *Beck-Chevalley square*, which is the natural notion of non-strict morphism of $\mathbf{V}'$-adjunctions.

Finally, we need to ask for T' to preserve the structure on computation types. Specifically, T' must preserve finite $\mathbf{V}'$-enriched products and coproducts, and $\mathbf{V}'$-powers and copowers indexed by objects in the range of S (such powers exist in $S_\star(\mathbf{C})$ since they are inherited from $\mathbf{C}$).

Definition 3. A *morphism of enriched-effect-calculus models* from $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ to $F' \dashv U'\colon \mathbf{C}' \to \mathbf{V}'$, is given by: a cartesian-closed functor $S\colon \mathbf{V} \to \mathbf{V}'$, which determines $S'\colon S_\star(\mathbf{V}) \to \mathbf{V}'$ as above, a fully faithful $\mathbf{V}'$-enriched functor $T'\colon S_\star(\mathbf{C}) \to \mathbf{C}'$, and a $\mathbf{V}'$-natural isomorphism $\alpha\colon F'\,S' \Rightarrow T'\,S_\star(F)$ such that (S', T', α) form a Beck-Chevalley square and T' preserves finite $\mathbf{V}'$-enriched products and coproducts, and $\mathbf{V}'$-powers and copowers indexed by objects in the range of S.

It can be shown that morphisms are closed under composition.

The main theorem of this section is that the syntactic models constructed in the proof of Theorem 2 enjoy an initiality property with respect to the above notion of morphism. Because the morphisms do not preserve structure strictly, canonical morphisms are unique only up to coherent natural isomorphism. Moreover, establishing that this property indeed holds turns out to be a surprisingly technical undertaking. For lack of space, the definition of coherent natural isomorphism and the proof of theorem below are omitted here.

Theorem 3. *Given an enriched-effect-calculus model $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ and families of objects, $\mathbf{V}[\![\alpha]\!]$ in $\mathbf{V}$ and $\mathbf{C}[\![\underline{\alpha}]\!]$ in $\mathbf{C}$, indexed by type constants, there exists a morphism of models from the syntactic model $F_{\mathrm{Syn}} \dashv U_{\mathrm{Syn}}\colon \mathbf{C}_{\mathrm{Syn}} \to \mathbf{V}_{\mathrm{Syn}}$ to $F \dashv U\colon \mathbf{C} \to \mathbf{V}$ that extends the given interpretation of type constants, and this morphism is unique up to coherent natural isomorphism.*

It should be pointed out that, given the way we have defined the syntactic model, the initiality property would not hold if morphisms were required to strictly preserve structure. The reason for this is that our economical syntax makes identifications that do not hold in arbitrary models. While this could be repaired by changing to Levy's syntax, we do not view it as a defect of our approach. The non-strict notion of morphism introduced in this section is the one that is useful in applications; for example, non-strict morphisms are needed in Section 6 below, and also in the companion paper [5].

6 Embedding an effect-calculus model in an enriched model

The purpose of this section is to establish that every effect-calculus model fully embeds in an enriched-effect-calculus model. It is this result that establishes that enriched-effect-calculus models are, in a precise sense, no less general than effect-calculus models. The important consequence is that the enriched effect calculus is compatible with arbitrary computational effects, whether commutative or not.

Theorem 4. *Every (small) effect-calculus model fully embeds in an enriched-effect-calculus model.*

Proof (outline). Given an effect-calculus model $F \dashv U \colon \mathbf{C} \to \mathbf{V}$, we need to fully embed this in an enriched-effect-calculus model $F' \dashv U' \colon \mathbf{C}' \to \mathbf{V}'$.

Since $\mathbf{V}$ and $\mathbf{C}$ are already $\widehat{\mathbf{V}}$-enriched, it is natural to define $\mathbf{V}' = \widehat{\mathbf{V}}$. Consider $\mathbf{D} = \mathbf{V}'[\mathbf{C}^{\mathrm{op}}, \mathbf{V}']$, i.e., the $\mathbf{V}'$-category of $\mathbf{V}'$-functors from $\mathbf{C}^{\mathrm{op}}$ to $\mathbf{V}'$. It is standard that $\mathbf{D}$ is complete and cocomplete as a $\mathbf{V}'$-category, and hence has all $\mathbf{V}'$-powers and copowers. Moreover, the enriched Yoneda functor $\mathbf{y}' \colon \mathbf{C} \to \mathbf{D}$ preserves all $\mathbf{V}'$-limits that exist in $\mathbf{V}$.

$\mathbf{C}'$ will be obtained as a subcategory of $\mathbf{D}$. To motivate its definition, we see what goes wrong if we try to use the whole of $\mathbf{D}$ for $\mathbf{C}'$. We first observe that any object I of $\mathbf{D}$ determines a $\mathbf{C}'$-enriched adjunction $(-) \cdot I \dashv \mathbf{D}(I, -) \colon \mathbf{D} \to \mathbf{C}'$. When I is chosen to be $\mathbf{y}'F1$, there is a $\mathbf{V}'$-natural isomorphism $\alpha' \colon \mathbf{y}\, U \Rightarrow \mathbf{D}(I, -)\, \mathbf{y}'$, because $U \cong \mathbf{C}(F1, -)$ and both $\mathbf{y}$ and $\mathbf{y}'$ preserve powers. However, we do not have a Beck-Chevalley square of adjunctions because the "mate" $\alpha \colon ((-) \cdot I)\, \mathbf{y} \Rightarrow \mathbf{y}'\, F$ of α', has the canonical maps $\psi_X \colon \mathbf{y}X \cdot \mathbf{y}'F1 \to \mathbf{y}'FX$ as its components. which are not generally isomorphisms.

To rectify the situation, we define $\mathbf{C}'$ to be the full sub-$\mathbf{V}'$-category of $\mathbf{D}$ on those objects orthogonal (in the $\mathbf{V}'$-enriched sense) to every ψ_X. One shows that all representables lie in $\mathbf{D}$ (because FX enjoys the universal property of $X \cdot F1$ in $\mathbf{C}$), and that $\mathbf{C}'$ has all $\mathbf{V}'$-limits, and these are calculated as in $\mathbf{D}$. Further, by [10, Theorem 6.5], $\mathbf{C}'$ is a ($\mathbf{V}'$-enriched) reflective subcategory of $\mathbf{D}$. It follows that $\mathbf{C}'$ has all $\mathbf{V}'$-colimits, in particular copowers and finite coproducts. Thus $(-) \cdot I \dashv \mathbf{C}'(I, -) \colon \mathbf{D} \to \mathbf{C}'$ is an enriched-effect-calculus model, where the copowers are calculated in $\mathbf{V}'$ (by reflecting copowers from $\mathbf{D}$). Moreover,

$$\begin{array}{ccc} \mathbf{C} & \xrightarrow{\quad \mathbf{y}' \quad} & \mathbf{C}' \\ F \uparrow \dashv \downarrow U & \simeq_{r(\alpha)} & (-) \cdot I \uparrow \dashv \downarrow \mathbf{V}'(I, -) \\ \mathbf{V} & \xrightarrow{\quad \mathbf{y} \quad} & \mathbf{V}' \end{array}$$

is now a Beck-Chevalley square of $\mathbf{V}'$-adjunctions, since the α components have been reflected to isomorphisms, and the α' components remain unchanged.

The above data determines the morphism we require. The functor $\mathbf{y} \colon \mathbf{V} \to \mathbf{V}'$ is cartesian closed. The $\mathbf{V}'$-functor $\mathbf{y}'$ is fully faithful (as in Definition 3,

this is a requirement for structure preservation). The Beck-Chevalley property means that the adjunction is preserved. Moreover, $\mathbf{y}'$ preserves all existing $\mathbf{V}'$-limits, in particular finite products and powers indexed by objects of $\mathbf{V}$ (qua representables). Thus we indeed have appropriate data for a morphism of models. The statement that this morphism is a full embedding amounts to $\mathbf{y}$ being fully faithful as a $\mathbf{V}'$-functor, which is standard. □

We apply the above embedding to complete the proof of Theorem 1.

Proof of Theorem 1.2 (outline). Suppose $\Gamma \mid \Delta \vdash s, t \colon \mathsf{A}$ are typable in the basic effect calculus and $\Gamma \mid \Delta \vdash s = t \colon \mathsf{A}$ holds in the enriched effect calculus, then $[\![s]\!] = [\![t]\!]$ holds in all models. Thus, for all models $F \dashv U \colon \mathbf{C} \to \mathbf{V}$ of the basic effect calculus, $[\![s]\!] = [\![t]\!]$ holds in the model $F' \dashv U' \colon \mathbf{C}' \to \mathbf{V}'$ given by Theorem 4. Since the latter model fully embeds the former, $[\![s]\!] = [\![t]\!]$ holds in every model of the basic effect calculus. By the completeness of such models, $\Gamma \mid \Delta \vdash s = t \colon \mathsf{A}$ holds in the basic effect calculus. □

References

1. A. Barber. *Linear Type Theories, Semantics and Action Calculi.* PhD Thesis, University of Edinburgh, 1997.
2. P.N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. *Proc. Computer Science Logic (CSL '94)*, Springer LNCS 933, 1995.
3. J. Berdine, P.W. O'Hearn, U. Reddy, H. Thielecke. Linear continuation-passing *Higher Order and Symbolic Computation*, 15:181–208, 2002.
4. P.N. Benton and P. Wadler. Linear logic, monads, and the lambda calculus. In *Proc. 11th Annual Symposium on Logic in Computer Science*, 1996.
5. J. Egger, R.E. Møgelberg and A. Simpson. Linearly-used continuations in an enriched effect calculus. In preparation, 2009.
6. A. Filinski. *Controlling Effects.* PhD thesis, School of Comp. Sci., CMU, 1996.
7. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
8. J.-Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.
9. M. Hasegawa. Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In *Proc. 6th International Symposium on Functional and Logic Programming*, Springer LNCS 2441, pages 167–182. 2002.
10. G.M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Math. Society Lecture Note Series.* Cambridge University Press, 1982.
11. P.B. Levy. *Call-by-push-value. A functional/imperative synthesis.* Semantic Structures in Computation, Springer, 2004.
12. P.B.Levy. Adjunction models for call-by-push-value with stacks. *Theory and Applications of Categories*, 14:75–110, 2005.
13. E. Moggi. Computational lambda-calculus and monads. In *Proc. 4th Annual Symposium on Logic in Computer Science*, pages 14–23, 1989.
14. E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
15. P.W. O'Hearn and J.C. Reynolds. From Algol to Polymorphic Linear Lambda-calculus. *Journal of the ACM*, 47:167–223, 2000.